Visual Basic Language Reference

GetObject Function (Visual Basic)

See Also Example

Collapse All Language Filter: Visual Basic

Returns a reference to an object provided by a COM component.

```
Public Function GetObject( _
    Optional ByVal PathName As String = Nothing, _
    Optional ByVal [Class] As String = Nothing _
) As Object
```

Parameters

Parameter	Description
PathName	Optional. **String**. The full path and name of the file containing the object to retrieve. If *PathName* is omitted, *Class* is required.
Class	Required if *PathName* is not supplied. **String**. A string representing the class of the object. The *Class* argument has the following syntax and parts: *appname*.*objecttype* **Parameter** / **Description** *appname* — Required. **String**. The name of the application providing the object. *objecttype* — Required. **String**. The type or class of object to create.

Exceptions

Exception type	Error number	Condition
Exception	429	No object with the specified path and file name exists.
FileNotFoundException	432	No object of the specified class type exists.

See the "Error number" column if you are upgrading Visual Basic 6.0 applications that use unstructured error handling. (You can compare the error number against the Number Property (Err Object).) However, when possible, you should consider replacing such error control with Structured Exception Handling Overview for Visual Basic.

Remarks

Use the **GetObject** function to load an instance of a COM component from a file. The following example illustrates this.

Copy Code

```
Dim CADObject As Object
CADObject = GetObject("C:\CAD\schema.cad")
```

When this code runs, the application associated with the specified *PathName* is started and the object in the specified file is activated.

Default Cases

If *PathName* is a zero-length string (""), **GetObject** returns a new object instance of the

specified class type. If the *PathName* argument is omitted, **GetObject** returns a currently active object of the class type specified in *Class*. If no object of the specified type exists, an error occurs.

Accessing a Subobject

Some applications allow you to activate a subobject associated with a file. To do this, add an exclamation point (**!**) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called `schema.cad`.

Copy Code

```
layerObject = GetObject("C:\CAD\schema.cad!Layer3")
```

Specifying a Class

If you do not specify the object's *Class*, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, can support more than one class of object. For example, a drawing might support three different types of objects: an `Application` object, a `Drawing` object, and a `Toolbar` object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional *Class* argument. The following example illustrates this.

Copy Code

```
Dim drawObj As Object
drawObj = GetObject("C:\Drawings\sample.drw", "Figment.Drawing")
```

In the preceding example, `Figment` is the name of a drawing application and `Drawing` is one of the object types it supports.

Using the Object

Once an object is activated, you refer to it in code using the object variable you declared. In the preceding example, you access properties and methods of the new object using the object variable `drawObj`. The following example illustrates this.

Copy Code

```
drawObj.Line(9, 90)
drawObj.InsertText(9, 100, "Hello, world.")
drawObj.SaveAs("C:\Drawings\sample.drw")
```

Note

Use the **GetObject** function when there is a current instance of the object or if you want to create the object with a file loaded. If there is no current instance, and you do not want the object started with a file loaded, use the CreateObject Function (Visual Basic).

If an object has registered itself as an ActiveX single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is called. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string (**""**) syntax, and it causes an error if the *PathName* argument is omitted. You cannot use **GetObject** to obtain a reference to a class created with Visual Basic.

Security Note

The **GetObject** function requires unmanaged code permission, which might affect its execution in partial-trust situations. For more information, see SecurityPermission and Code Access Permissions.

Example

The following example uses the **GetObject** function to obtain a reference to a specific Microsoft Excel worksheet (`excelObj`). It uses the worksheet's **Application** property to make Excel visible, to close it, and to perform other actions. Using two API calls, the `detectExcel` procedure looks for Excel, and if it is running, enters it in the Running Object table. The first call to **GetObject** causes an error if Excel is not already running, which in this example causes the `excelWasNotRunning` flag to be set to **True**. The second call to **GetObject** specifies a file to open. If Excel is not already running, the second call starts it and returns a reference to the worksheet represented by the specified file, `test.xls`. The file must exist in the specified location; otherwise, Visual Basic throws a **FileNotFoundException**. Next, the example code makes both Excel and the window containing the specified worksheet visible.

This example requires **Option Strict Off** because it uses late binding, where objects are assigned to variables of type **Object**. You can specify **Option Strict On** and declare objects of specific object types if you add a reference to the Excel type library from the **COM** tab of the **Add Reference** dialog box of the **Project** menu in Visual Studio.

Visual Basic Copy Code

```
' Add Option Strict Off to the top of your program.
Option Strict Off
```

Visual Basic Copy Code

```
' Declare necessary API routines.
Declare Function findWindow Lib "user32.dll" Alias _
    "FindWindowA" (ByVal lpClassName As String, _
    ByVal lpWindowName As Long) As Long
Declare Function sendMessage Lib "user32.dll" Alias _
    "SendMessageA" (ByVal hWnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, ByVal lParam As Long) As Long
```

Visual Basic Copy Code

```
Sub getExcel()
    Dim excelObj As Object
    Dim excelWasNotRunning As Boolean
    ' Test to see if a copy of Excel is already running.
    On Error Resume Next
    ' GetObject called without the first argument returns a
    ' reference to an instance of the application. If the
    ' application is not already running, an error occurs.
    excelObj = GetObject(, "Excel.Application")
    If Err().Number <> 0 Then excelWasNotRunning = True
    Err().Clear()
    ' If Excel is running, enter it into the Running Object table.
    detectExcel()
    ' Set the object variable to refer to the file you want to use.
    excelObj = GetObject("c:\vb\test.xls")
    ' Show Excel through its Application property. Then show the
    ' window containing the file, using the Windows collection of
    ' the excelObj object reference.
    excelObj.Application.Visible = True
    excelObj.Parent.Windows(1).Visible = True
    ' Insert code to manipulate the test.xls file here.
End Sub
```

Visual Basic Copy Code

```
Sub detectExcel()
    ' Procedure detects a running Excel and registers it.
    Const WM_USER As Long = 1024
    Dim hWnd As Long
    ' If Excel is running, this API call returns its handle.
    hWnd = findWindow("XLMAIN", 0)
    If hWnd = 0 Then
```

```
            ' 0 means Excel is not running.
            Exit Sub
        Else
            ' Excel is running, so use the sendMessage API function
            ' to enter it in the Running Object table.
            sendMessage(hWnd, WM_USER + 18, 0, 0)
        End If
    End Sub
```

When you call the `getExcel` function, a check is made to see if Excel is already running. If it is not, then an instance is created.

Security Note

For simplicity, the preceding example assumes that any window called `XLMAIN` belongs to an instance of Microsoft Excel. If another object, possibly launched by illicit tampering, created a window with that name, it would receive all the messages you intended for Excel. In an application to be used for production, you should include some more rigorous testing to verify that `XLMAIN` really belongs to Excel.

Smart Device Developer Notes

This function is not supported.

Requirements

Namespace: Microsoft.VisualBasic

Module: Interaction

Assembly: Visual Basic Runtime Library (in Microsoft.VisualBasic.dll)

See Also

Reference

CreateObject Function (Visual Basic)
Declare Statement
Option Strict Statement
Exception
FileNotFoundException